

UNIT22: SKINNED MESH - BONE INFLUENCE

【学習要項】

□Bone influences

【演習手順】

1. 3つのボーンを持つ FBX ファイル (cube.004.fbx) をロードし各ボーンの頂点に対する影響度を確認する
※cube.004.fbx は右手系・Z 軸アップ・メートル単位・三角形化なし
2. 頂点構造体(vertex)にボーン番号とウェイト値を追加する

```
*1: static const int MAX_BONE_INFLUENCES{ 4 };
2: struct vertex
3: {
4:     DirectX::XMFLOAT3 position;
5:     DirectX::XMFLOAT3 normal;
6:     DirectX::XMFLOAT2 texcoord;
*7:     float bone_weights[MAX_BONE_INFLUENCES]{ 1, 0, 0, 0 };
*8:     uint32_t bone_indices[MAX_BONE_INFLUENCES]{};
9: };
```

3. skinned_mes.cpp のグローバルスコープにボーン影響度を表現する構造体を定義する
※bone_influence 構造体の1つのインスタンスは1つ頂点が影響を受けるボーン番号とその重みを表現する
※1つ頂点は複数のボーンから影響を受ける場合があるので可変長配列で表現している

```
1: struct bone_influence
2: {
3:     uint32_t bone_index;
4:     float bone_weight;
5: };
6: using bone_influences_per_control_point = std::vector<bone_influence>;
```

4. skinned_mesh.cpp のグローバルスコープにボーン影響度を FBX データから取得する関数を定義する

```
1: void fetch_bone_influences(const FbxMesh* fbx_mesh,
2:     std::vector<bone_influences_per_control_point>& bone_influences)
3: {
4:     const int control_points_count{ fbx_mesh->GetControlPointsCount() };
5:     bone_influences.resize(control_points_count);
6:
7:     const int skin_count{ fbx_mesh->GetDeformerCount(FbxDeformer::eSkin) };
8:     for (int skin_index = 0; skin_index < skin_count; ++skin_index)
9:     {
10:         const FbxSkin* fbx_skin
11:             { static_cast<FbxSkin*>(fbx_mesh->GetDeformer(skin_index, FbxDeformer::eSkin)) };
12:
13:         const int cluster_count{ fbx_skin->GetClusterCount() };
14:         for (int cluster_index = 0; cluster_index < cluster_count; ++cluster_index)
15:         {
16:             const FbxCluster* fbx_cluster{ fbx_skin->GetCluster(cluster_index) };
17:
18:             const int control_point_indices_count{ fbx_cluster->GetControlPointIndicesCount() };
19:             for (int control_point_indices_index = 0; control_point_indices_index < control_point_indices_count;
20:                 ++control_point_indices_index)
21:             {
22:                 int control_point_index{ fbx_cluster->GetControlPointIndices()[control_point_indices_index] };
23:                 double control_point_weight
24:                     { fbx_cluster->GetControlPointWeights()[control_point_indices_index] };
25:                 bone_influence& bone_influence{ bone_influences.at(control_point_index).emplace_back() };
26:                 bone_influence.bone_index = static_cast<uint32_t>(cluster_index);
27:                 bone_influence.bone_weight = static_cast<float>(control_point_weight);
28:             }
29:         }
30:     }
31: }
```

5. 頂点が影響を受けるボーン番号とウェイト値を頂点構造体(vertex)のメンバ変数にセットする
※影響を受けるボーンは最大4つまでとする

UNIT22: SKINNED MESH - BONE INFLUENCE

※下記コードは影響を受けるボーン数が4つを超える場合は、それ以降の影響度を無視している（改善しなさい）

```
1: void skinned_mesh::fetch_meshes(FbxScene* fbx_scene, std::vector<mesh>& meshes)
2: {
3:     for (const scene::node& node : scene_view.nodes)
4:     {
5:         if (node.attribute != FbxNodeAttribute::EType::eMesh)
6:         {
7:             continue;
8:         }
9:
10:        FbxNode* fbx_node{ fbx_scene->FindNodeByName(node.name.c_str()) };
11:        FbxMesh* fbx_mesh{ fbx_node->GetMesh() };
12:
13:        mesh& mesh{ meshes.emplace_back() };
14:        mesh.unique_id = fbx_mesh->GetNode()->GetUniqueID();
15:        mesh.name = fbx_mesh->GetNode()->GetName();
16:        mesh.node_index = scene_view.indexof(mesh.unique_id);
17:        mesh.default_global_transform = to_xmfloat4x4(fbx_mesh->GetNode()->EvaluateGlobalTransform());
18:
19:        std::vector<bone_influences_per_control_point> bone_influences;
20:        fetch_bone_influences(fbx_mesh, bone_influences);
21:
22:        :
23:        :
24:        :
25:
26:        for (int polygon_index = 0; polygon_index < polygon_count; ++polygon_index)
27:        {
28:            const int material_index{ material_count > 0 ?
29:                fbx_mesh->GetElementMaterial()->GetIndexArray().GetAt(polygon_index) : 0 };
30:            mesh::subset& subset{ subsets.at(material_index) };
31:            const uint32_t offset{ subset.start_index_location + subset.index_count };
32:
33:            for (int position_in_polygon = 0; position_in_polygon < 3; ++position_in_polygon)
34:            {
35:                const int vertex_index{ polygon_index * 3 + position_in_polygon };
36:
37:                vertex vertex;
38:                const int polygon_vertex
39:                    { fbx_mesh->GetPolygonVertex(polygon_index, position_in_polygon) };
40:                vertex.position.x = static_cast<float>(control_points[polygon_vertex][0]);
41:                vertex.position.y = static_cast<float>(control_points[polygon_vertex][1]);
42:                vertex.position.z = static_cast<float>(control_points[polygon_vertex][2]);
43:
44:                const bone_influences_per_control_point& influences_per_control_point
45:                    { bone_influences.at(polygon_vertex) };
46:                for (size_t influence_index = 0; influence_index < influences_per_control_point.size();
47:                    ++influence_index)
48:                {
49:                    if (influence_index < MAX_BONE_INFLUENCES)
50:                    {
51:                        vertex.bone_weights[influence_index] =
52:                            influences_per_control_point.at(influence_index).bone_weight;
53:                        vertex.bone_indices[influence_index] =
54:                            influences_per_control_point.at(influence_index).bone_index;
55:                    }
56:                }
57:
58:                :
59:                :
60:                :
61:
62:            }
63:        }
64:    }
```

UNIT22: SKINNED MESH - BONE INFLUENCE

```
65: }
```

6. `skinned_mesh` の `create_com_objects` メンバ関数でインプットレイアウト定義を変更する

```
1: D3D11_INPUT_ELEMENT_DESC input_element_desc[]
2: {
3:     { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT, 0, 0 },
4:     { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT, 0, 0 },
5:     { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT, 0, 0 },
*6:     { "WEIGHTS", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT, 0, 0 },
*7:     { "BONES", 0, DXGI_FORMAT_R32G32B32A32_UINT, 0, D3D11_APPEND_ALIGNED_ELEMENT, 0, 0 },
8: };
```

7. 頂点シェーダの入力構造体(VS_IN)を変更する

```
1: struct VS_IN
2: {
3:     float4 position : POSITION;
4:     float4 normal : NORMAL;
5:     float2 texcoord : TEXCOORD;
*6:     float4 bone_weights : WEIGHTS;
*7:     uint4 bone_indices : BONES;
8: };
```

8. 頂点シェーダ(`skinned_mesh_vs.hlsl`)にボーン影響度を確認するテストコードを追加する

※動作確認後`#if-#endif`ディレクティブのコードは無効にすること(15:-26:行目)

```
1: VS_OUT main(VS_IN vin)
2: {
3:     VS_OUT vout;
4:     vout.position = mul(vin.position, mul(world, view_projection));
5:
6:     vout.world_position = mul(vin.position, world);
7:     vin.normal.w = 0;
8:     vout.world_normal = normalize(mul(vin.normal, world));
9:
10:    vout.texcoord = vin.texcoord;
11:
*12: #if 0
*13:     vout.color = material_color;
*14: #else
*15:     vout.color = 0;
*16:     const float4 bone_colors[4] = {
*17:         {1, 0, 0, 1},
*18:         {0, 1, 0, 1},
*19:         {0, 0, 1, 1},
*20:         {1, 1, 1, 1},
*21:     };
*22:     for (int bone_index = 0; bone_index < 4; ++bone_index)
*23:     {
*24:         vout.color += bone_colors[vin.bone_indices[bone_index] % 4]
*25:             * vin.bone_weights[bone_index];
*26:     }
*27: #endif
28:
29:     return vout;
30: }
```

9. 実行し、テスト描画を確認する (スクリーンショット画像: `cube.004.0.png`)

【評価項目】

☐ ボーン影響度のテスト描画 (`cube.004.fbx`)